目录

[安装过程 1](#_Toc476472142)

[添加远程仓库 1](#_Toc476472143)

[添加远程仓库、 3](#_Toc476472144)

[Git的使用教程（四）查看日志和版本回退 4](#_Toc476472145)

[git fetch 的简单用法:更新远程代码到本地仓库 11](#_Toc476472146)

[git branch（分支命令的使用） 12](#_Toc476472147)

安装过程

安装完成后，还需要最后一步设置，在命令行输入：

$ git config --global user.name "Your Name"

$ git config --global user.email "bingwei2642@qq.com"

因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址。你也许会担心，如果有人故意冒充别人怎么办？这个不必担心，首先我们相信大家都是善良无知的群众，其次，真的有冒充的也是有办法可查的。

注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

添加远程仓库

第1步：创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

$ ssh-keygen -t rsa -C "bingwei2642@qq.com"

你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，由于这个Key也不是用于军事目的，所以也无需设置密码。

如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

第2步：登陆GitHub，打开“Account settings”，“SSH Keys”页面：

然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容：



点“Add Key”，你就应该看到已经添加的Key：



为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送。

当然，GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。

最后友情提示，在GitHub上免费托管的Git仓库，任何人都可以看到喔（但只有你自己才能改）。所以，不要把敏感信息放进去。

如果你不想让别人看到Git库，有两个办法，一个是交点保护费，让GitHub把公开的仓库变成私有的，这样别人就看不见了（不可读更不可写）。另一个办法是自己动手，搭一个Git服务器，因为是你自己的Git服务器，所以别人也是看不见的。这个方法我们后面会讲到的，相当简单，公司内部开发必备。

确保你拥有一个GitHub账号后，我们就即将开始远程仓库的学习。

添加远程仓库、

现在，我们根据GitHub的提示，在本地的learngit仓库下运行命令：

$ git remote add origin git@github.com:michaelliao/learngit.git

请千万注意，把上面的michaelliao替换成你自己的GitHub账户名，否则，你在本地关联的就是我的远程库，关联没有问题，但是你以后推送是推不上去的，因为你的SSH Key公钥不在我的账户列表中。

添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。

下一步，就可以把本地库的所有内容推送到远程库上：

$ git push -u origin master

Counting objects: 19, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (19/19), done.

Writing objects: 100% (19/19), 13.73 KiB, done.

Total 23 (delta 6), reused 0 (delta 0)

To git@github.com:michaelliao/learngit.git

\* [new branch] master -> master

Branch master **set** up **to** track remote branch master **from** origin.

# [Git的使用教程（四）查看日志和版本回退](http://blog.csdn.net/a7442358/article/details/50384293)

标签： [git](http://www.csdn.net/tag/git)

2016-02-13 08:37 510人阅读 [评论](http://blog.csdn.net/a7442358/article/details/50384293#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/a7442358/article/details/50384293#report)

本文章已收录于：

[[](http://lib.csdn.net/base/28?source=blogtop) **Git知识库**](http://lib.csdn.net/base/28?source=blogtop)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

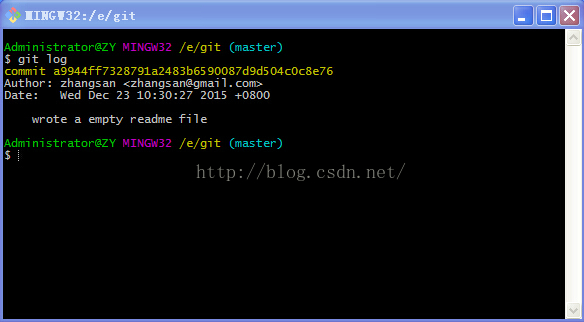
git（6） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

    我们使用任何的版本管理工具，原因无外乎就是想要记录文件新增、修改、删除等相关操作，当我们修改文件出现错误的时候，可以及时回到没有出错的“备份”文件。当然[**Git**](http://lib.csdn.net/base/git)作为优秀的版本管理工具这样的功能当然不能少，下面我们就来看下与版本管理相关的一些命令吧。

**命令：git log**

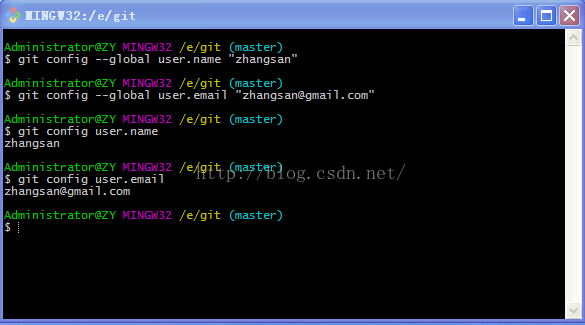
    git log可以记录我们git commit操作（不会记录git add操作，如果你没有进行过一次git commit，那么git log的时候什么都不会显示），每次git commit都会产生一条记录，每条记录代表一次commit，一条记录共有四部分组成。



**commit**：是一个由SHA1计算出来的非常大的数字，用十六进制表示，是此次操作的唯一标识符（每次commit的时候该值不会重复），我们可以通过它准确的找到此次提交操作，为恢复到该版本打下坚实的基础。

**Author**： 提交该版本的作者以及该作者的邮箱

    在多人协作中，每个人都有可能对文件做出一些调整，如何区分是什么人做了这些调整呢，这就用到了Author。我们可以使用git config user.name和git config user.email来获取当前用户名和用户邮箱，如果没有设置或是需要调整，可以通过git config --global user.name "new name"来进行调整。设置Author很有必要**，最好在git init之后就立刻设置**，这样在你进行git commit的时候就能记录你设置的Author信息了。



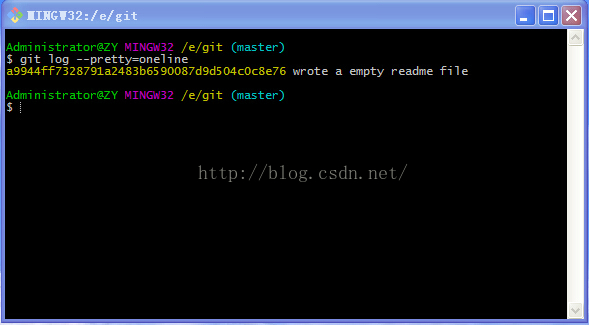
**Date**：提交的日期。

**最后**：最后一行就是关于此次提交的说明，也就是我们git commit -m的内容了，这可以让我们更加清楚为什么做此次提交操作，为以后的版本恢复打下坚实基础。

    直接使用git log，会展示每次提交的详细的信息，但是当commit信息过多，亦或者我们不想要这么详细的信息的时候，此时我们就可以使用git log的另一种用法。

**[plain]** [view plain](http://blog.csdn.net/a7442358/article/details/50384293) [copy](http://blog.csdn.net/a7442358/article/details/50384293)

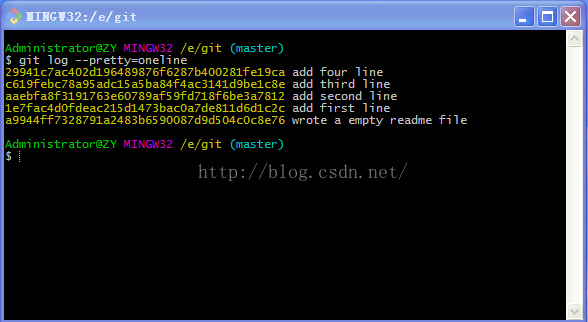
1. git log --pretty=oneline



变少了有某有，这样一屏下来就可以展示更多的commit信息了。

**命令：git reset**

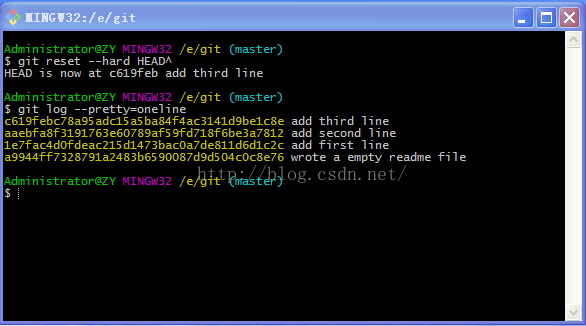
    git log只是帮我们记录了每次的提交操作，我们想要进行不同的版本切换，还需要用到git reset命令。在此之前，我们先多进行几次提交操作。准备工作做好了，正式开始吧。



**方法一：**

**[plain]** [view plain](http://blog.csdn.net/a7442358/article/details/50384293) [copy](http://blog.csdn.net/a7442358/article/details/50384293)

1. git reset --hard HEAD^

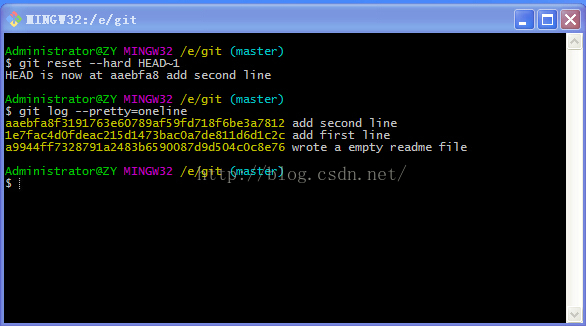


git reset是版本回退命令，--hard是回退命令的一个参数，暂时不用深究它具体代表什么含义，只要记住这样操作可以达到我们回退的目的即可。前面的部分我们暂且把它当做是实现版本回退的一种固定格式，我们重点来说下HEAD^。在Git中，HEAD代表的是当前版本，拿上述例子来说就是"add four line"，而加个^，就表示当前版本之前的版本，就是例子中“add third line”，每加一个^就表示更前的一个版本，例如HEAD^^就表示“add second line”这个版本，如果想要回到100个版本前，就需要100个^，当然回退的版本太靠前的话，光写^也是个麻烦事，还有，Git提供了另一种写法解决了这个问题。

**方法二：**

**[plain]** [view plain](http://blog.csdn.net/a7442358/article/details/50384293) [copy](http://blog.csdn.net/a7442358/article/details/50384293)

1. git reset --hard HEAD~1



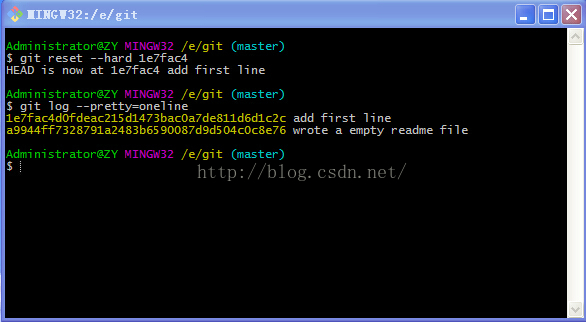
把HEAD^换成了HEAD~1，其中1代表回退几个版本，如果需要回退到10个之前的版本，只需要把HEAD~1改为HEAD~10即可，是不是比10个^方便了好多呢。看起来是方便了点，但是仍有不足，如果想要回退到某个版本的时候，岂不是还要查一下这个版本和当前版本之间的差。Git也考虑到了这个问题，因此提供了第三种方法供我们使用。

**方法三：**

**[plain]** [view plain](http://blog.csdn.net/a7442358/article/details/50384293) [copy](http://blog.csdn.net/a7442358/article/details/50384293)

[在CODE上查看代码片](https://code.csdn.net/snippets/1577541)

1. git reset --hard 1e7fac4



还记得刚才git log中提到的commit吗，此时我们就是把HEAD换成了commit。正是由于commit字串的唯一性，我们才能快速准确的回退到我们需要的版本，而不需要查找当前版本和切换版本之间的差值。关于--hard后面的这个字串，相信不少朋友还有疑问，下面小编就一一解答。

**疑问一：该字串从哪里来的？**

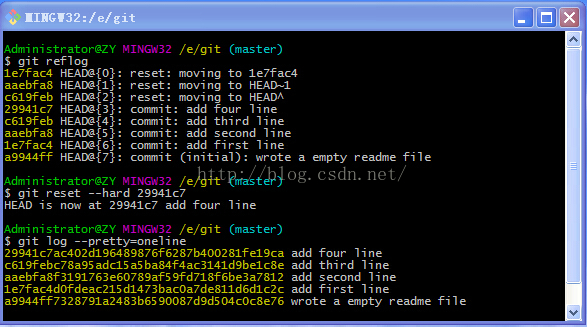
    每次git commit的时候Git都会生成这样一个字串方便做版本管理，可以通过git log来查看每个版本具体commit字串值。

**疑问二：进行版本回退的时候如何填写该字串？**

    其实这个并没有硬性的规定，经过小编测试，这个字串至少是4位，也就是说我们至少需要取commit字串的前4位（4位以下会报错）。其实输入的这个字串是为了确保Git能够准确的找到对应的版本，如果版本中有前4位重复的，那么我们就需要多增加一位，确保Git不会找错了。那么多少位适合呢？如果使用Git时间久了，就会发现，很多操作使用到这个commit的时候都会把它缩略为前7位，因此，我们也可以沿用，使用前7位作为默认位数，当然，如果前7位仍不唯一，就需要输入前8位了。

**命令：git reflog**

    一次又一次的回退操作已经使得我们忘乎所以，不知不觉已经快回退到最初的版本了，但这时候我们又怀念起了新版本的好，想要回去，可是git log中已经没有新版本的任何足迹了，我们还回的去吗？答案是肯定的，凡走过必留下痕迹，Git也同样为我们留下了蛛丝马迹。



git reflog是Git提供给我们记录命令历史，commit和reset都在被记录之列。通过git reflog，我们发现了最后一次提交的commit的字串，有了它我们是不是可以找回最新的版本呢？抱着怀疑的态度我们进行了尝试，再看下git log，哇塞，居然回到了进行回退前的状态，这样一来我们岂不是可以自由徜徉在各个版本之间了。

# [git fetch 的简单用法:更新远程代码到本地仓库](http://blog.csdn.net/hanfengzqh/article/details/52790653)

git fetch origin master:tmp

git diff tmp

git merge tmp  
git branch -d tmp  
      
    从远程获取最新的版本到本地的tmp分支上  
   之后再进行比较合并  
  
2. git pull：相当于是从远程获取最新版本并merge到本地  
     
git pull origin master  
  
  
    上述命令其实相当于git fetch 和 git merge  
    在实际使用中，git fetch更安全一些  
   因为在merge前，我们可以查看更新情况，然后再决定是否合并

# [git branch（分支命令的使用）](http://hbiao68.iteye.com/blog/2055493)

**博客分类：**

* [代码管理 & svn & git](http://hbiao68.iteye.com/category/198306)

查看分支

git branch

或者

git branch -v

A) 创建分支

git branch mystudygit1.0

B) 切换分支

git checkout mystudygit1.0

C) 删除分支

git branch -d mystudygit1.0  //如果该分支没有合并到主分支会报错

或者

git branch -D mystudygit1.0   //强制删除

D) 分支合并

比如，如果要将开发中的分支（develop），合并到稳定分支（master），

          首先切换的master分支：git checkout master。

然后执行合并操作：git merge develop。

如果有冲突，会提示你，调用git status查看冲突文件。

解决冲突，然后调用git add或git rm将解决后的文件暂存。

所有冲突解决后，git commit 提交更改。

例如：将acc2f69提交合并到当前分支

git merge acc2f69

E）合并

git如何clone 远程github中的分支？

git clone -b release\_branch https://github.com/jetty/